Implementation of A* Algorithm

Srinidhi.V
18C102

Ex.No.2
03.2021

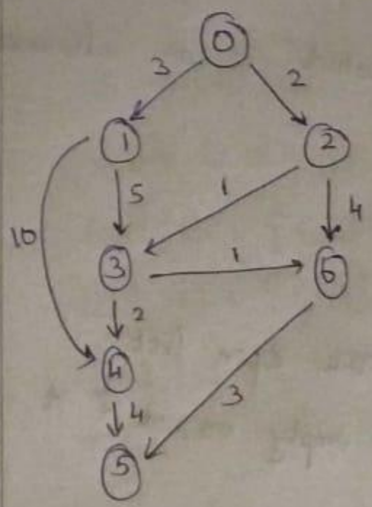
Aim:

To implement A* algorithm which is an informed search algorithm.

Algorithm:

* Start
* Place the starting node in the open list.
* Check if the open list is empty or not if it, then return failure.
* Select the node from the open list which has the smallest value of evaluation function (g+h). If node is a goal node, then return success and stop otherwise.
* Explore and generate all successors and for each successor check if its already in open or closed list and if not complete evaluation function and store it in open list.
* Else if node n is already in open and closed, then it should be attached.
* If the open list is empty, then return failure.
* End

Sample Input And Output:



Heuristic Value:

	h(n)
0	7
1	9
2	5
3	4
4	2
5	0
6	3

$f(n) = g(n) + h(n)$ is used in exploring nodes

Start node → 0 Goal node → 5

Node	h(n)	g(n)	f(n)
0	7	0	7
1	9	3	12
2	5	2	7
3	4	3	7
4	2	5	7
5	0	9	9
6	3	6	9

Path taken from 0 to reach 5:

$0 \xrightarrow{2} 2 \xrightarrow{1} 3 \xrightarrow{1} 6 \xrightarrow{3} 5$

Total path cost: 7

Program:

```
import heapq as hq

def makeGraph():
    global adj, heur
    adj['S'] = [('A', 3), ('B', 1), ('C', 5)]
    adj['A'] = [('G1', 10), ('E', 7)]
    adj['B'] = [('C', 2), ('F', 2)]
    adj['C'] = [('G3', 11)]
    adj['D'] = [('S', 6), ('B', 4), ('G2', 5)]
    adj['E'] = [('G1', 2)]
    adj['F'] = [('D', 1)]

    heur = {'S': 8, 'A': 9, 'B': 1, 'C': 3, 'D': 4, 'E': 1, 'F': 5, 'G1': 0, 'G2': 0, 'G3': 0 }


def a_star(src, adj, heur, goals):
    pq, expanded_nodes, expanded = [], dict(), set()
    hq.heapify(pq)
    hq.heappush(pq, (heur[src] + 0, src, heur[src], 0, '$')) #(h_val + g_val, curr_node, h_val,
g_val, parent)

    while(len(pq) != 0):
        curr = hq.heappop(pq)
        curr_node = curr[1]
        edge_cost = curr[3]
        parent = curr[4]

        if(curr_node in goals):
            expanded_nodes[curr_node] = parent
            break

        if(curr_node not in expanded):
            expanded.add(curr_node)
            expanded_nodes[curr_node] = parent
        else:
            continue

        for node in adj[curr_node]:
            dest_node, dest_cost = node
            g_val = edge_cost + dest_cost
            h_val = heur[dest_node]
            total_cost = g_val + h_val
```

```
            hq.heappush(pq, (total_cost, dest_node, h_val, g_val, curr_node))

    path = []
    path = findPath(expanded_nodes, src)
    return path

def findPath(exp_nodes, start):
    child = list(exp_nodes.keys())[-1] # Get last key value from dict which is the goal
    parent = exp_nodes[child]
    res = []

    while(parent != '$'):
        res.append(child)
        child = parent
        parent = exp_nodes[child]

    res.append(child)
    return res

def displayResult(path, adj, heur, start, goals):
    res, parent, goal, index = [], path[-1], path[0], -2
    curr_cost = heur[parent]
    res.append((parent, curr_cost))

    while(parent != goal):
        child = path[index]
        index -= 1
        for node in adj[parent]:
            if(node[0] == child):
                curr_cost += node[1] + heur[child]
                res.append((child, curr_cost))
                parent = child
                break

    print("\nAdjacency List : ", adj)
    print("\nHeuristics : ", heur)
    print("\nStarting node : ", start)
    print("\nGoals : ", goals)
    print("\nPath from Starting to Goal Node :")
    for node in res:
        print(f"{node[0]} ({node[1]}) -> ", end = " ")
    print("GOAL \nTotal Cost = ", curr_cost)
```

```
if __name__ == "__main__":
    adj, heur = dict(), dict()
    makeGraph()
    goals, start = ["G1", "G2", "G3"], "S"
    path = a_star(start, adj, heur, goals)
    displayResult(path, adj, heur, start, goals)
```

Output:

```
Adjacency List :  {'S': [('A', 3), ('B', 1), ('C', 5)], 'A': [('G1', 10), ('E', 7)], 'B': [('C', 2), ('F', 2)], 'C': [('G3', 11)], 'D': [('S', 6), ('B', 4), ('G2', 5)], 'E': [('G1', 2)], 'F': [('D', 1)]}

Heuristics :  {'S': 8, 'A': 9, 'B': 1, 'C': 3, 'D': 4, 'E': 1, 'F': 5, 'G1': 0, 'G2': 0, 'G3': 0}

Starting node :  S

Goals :  ['G1', 'G2', 'G3']

Path from Starting to Goal Node :
S (8) ->  B (10) ->  F (17) ->  D (22) ->  G2 (27) ->  GOAL
Total Cost =  27
```

Result:

Thus, the program for A* is implemented successfully.